



# **FMS R1B2 High Level Design**

**Contract DBM-9713-NMS  
TSR # 9803444  
Document # M303-DS-002R0**

**June 9, 2000**

**By**

**Computer Sciences Corporation and PB Farradyne Inc**





# Table of Contents

---

<b>1</b>	<b>Introduction.....</b>	<b>1-1</b>
1.1	Purpose.....	1-1
1.2	Objectives.....	1-1
1.3	Scope.....	1-1
1.4	Design Process .....	1-1
1.5	Design Tools.....	1-2
1.6	Work Products.....	1-2
<b>2</b>	<b>Software Architecture.....</b>	<b>2-3</b>
2.1	Communications Servers (FMS Remote PC) .....	2-3
2.2	Port Manager.....	2-3
2.3	Device Protocol Handlers .....	2-4
2.4	CHART II Interface.....	2-5
2.5	High Availability .....	2-5
2.5.1	Device object deployment .....	2-6
2.6	Failures and Health Status .....	2-7
2.7	Database Usage.....	2-8
2.8	ITS National Standards Approach .....	2-8
2.9	R1B1 Problem Resolution .....	2-8
2.10	Future Enhancements .....	2-9
2.10.1	Currently Planned Enhancements.....	2-9
2.10.2	Other Enhancements.....	2-10
<b>3</b>	<b>Models .....</b>	<b>3-1</b>
3.1	Class Diagrams .....	3-1
3.1.1	FMSSubsystem (Class Diagram).....	3-1
3.1.2	FMSProtocolHandlers (Class Diagram).....	3-4
3.2	Sequence Diagrams .....	3-6
3.2.1	Typical FMS Usage – Set DMS Message .....	3-6

3.2.2	Multiple Commands on Single Connection.....	3-7
3.2.3	Fault Tolerance Scenario .....	3-8
3.2.4	Port Leak Prevention .....	3-9
<b>4</b>	<b>Packaging.....</b>	<b>4-1</b>
<b>5</b>	<b>Deployment .....</b>	<b>5-1</b>
	<b>Bibliography .....</b>	<b>1</b>
	<b>Acronyms .....</b>	<b>1</b>
	<b>Glossary .....</b>	<b>1</b>
	<b>Appendix A: CORBA Information .....</b>	<b>1</b>
	<b>CORBA .....</b>	<b>1</b>
	<b>CORBA Services .....</b>	<b>1</b>
	CORBA Event Service.....	1
	CORBA Trading Service .....	1

## Table of Figures

Figure 1.	FMS R1B1 Hardware Deployment (Deployment Diagram) .....	2-5
Figure 2.	Deployment for Lower Bandwidth Device Objects (Deployment Diagram).....	2-6
Figure 3.	Deployment for Higher Bandwidth Device Objects (Deployment Diagram) .....	2-7
Figure 4.	FMSSubsystem (Class Diagram).....	3-1
Figure 5.	FMSProtocolHandlers (Class Diagram) .....	3-4
Figure 6.	Typical FMS Usage – Set DMS Message (Sequence Diagram) .....	3-6
Figure 7.	Multiple Commands On Single Connection (Sequence Diagram).....	3-7
Figure 8.	Fault Tolerance Scenario (Sequence Diagram) .....	3-8
Figure 9.	FMSSubsystem:PortLeakPrevention (Sequence Diagram) .....	3-9
Figure 10.	FMSPackaging (Class Diagram) .....	4-1
Figure 11.	FMSDeployment (Deployment Diagram) .....	5-1

# 1 Introduction

---

## 1.1 Purpose

This document describes the high level design of the FMS software for Release 1, Build 2. This design is driven by the requirements of CHART II, the user of the FMS subsystem, as stated in document M361-RS-002R2, “*CHART II System Requirements Specification*”.

## 1.2 Objectives

The main objective of this design is to provide software developers with a framework in which to provide detailed design and implementation of the software components that comprise the FMS subsystem.

This design also serves to provide documentation to those outside of the software development community to show how communications specific requirements of CHART II are being accounted for in the software design.

## 1.3 Scope

This design is limited to Release 1, Build 2 of the FMS subsystem and the communication specific requirements as stated in the aforementioned requirements document (Section 1.1).

## 1.4 Design Process

Object oriented analysis and design techniques were used in creating this design. As such, much of the design is documented using diagrams that conform to the Unified Modeling Language (UML), a de facto standard for diagramming object-oriented designs.

In addition to being object oriented, this design incorporates distributed object techniques, which allow for great flexibility and scalability of the system. In a distributed object system, objects can be deployed in servers throughout the network. This design addresses the partitioning of object types into specific server applications for this release.

The design process is very iterative: each step can possibly cause changes to previous steps. Listed below is the process that was used to create the work products contained in this document:

- The team utilized the use case diagrams defined in the CHART II R1B2 High Level Design and identified uses that require field communications.
- A straw man class diagram was created with major entities evident in the use cases being listed as possible classes in the system. High-level relationships between the classes were discovered and documented on the class diagram.
- Sequence diagrams were created for key uses of the system, showing how the classes on the class diagram would be used to perform the use case. This often involved changes to the class diagram, such as adding classes, moving responsibilities between classes, or adding operations to a class. Sometimes the changes affected other sequence diagrams as well.
- After the process of creating sequence diagrams and associated changes to the class diagram, internal reviews were used to resolve remaining issues.

- The design was broken down into packages, grouping classes with a high amount of dependency together.
- Deployment diagrams were created to show various deployment options that exist with the FMS and CHART II objects.

## **1.5 Design Tools**

The work products contained within this design are extracted from the COOL:JEX design tool. Within this tool, the design is contained in the Chart II project, R1B2 configuration, Analysis phase, system version FMSHighLevel.

## **1.6 Work Products**

This design contains the following work products:

- UML Class diagrams, showing the high level software objects which will allow the system to accommodate the uses of the system described in the Use Case diagrams.
- UML Sequence diagrams showing how the classes interact to accomplish a use of the system.
- A UML Package diagram, showing how the classes are broken up into manageable software packages.
- A UML Deployment diagram, showing which servers will serve each class of objects and where servers and GUIs will be deployed.

## 2 Software Architecture

---

The FMS software architecture provides generic distributed communication services which are used by device specific protocol handlers to provide access to field devices such as Dynamic Message Signs, Highway Advisory Radio, and others. The Common Object Request Broker Architecture (CORBA) is used as a base for the FMS architecture. Background information on CORBA can be found in appendix B.

The sections below discuss specific elements of the architecture and software components that comprise the FMS subsystem.

### 2.1 Communications Servers (FMS Remote PC)

A communication server is a PC that is outfitted with one or more pieces of communications hardware, such as ISDN modems, POTS modems, and/or telephony boards used to communicate with field devices. FMS software is run on a communication server to provide the management of these communications resources to allow them to be shared among separate software applications and also to provide remote access to these resources via a network. Because the FMS software can be distributed, deployment options exist to allow applications that control field devices to be highly tolerant to faults in the communications servers, their hardware, and the communications infrastructure itself.

### 2.2 Port Manager

The FMS software that manages access to communications resources is a PortManager. A PortManager is configured specifically for the hardware that it will manage. The communications resources are modeled in software as Port objects. Specific types of port objects exist for each type of communications resource that is supported, which currently includes ISDN and POTS modems.

Upon startup of the FMS software, a PortManager object is created and published to the CORBA trading service, making it available for discovery and use by other applications. The PortManager creates port objects to represent each of the physical communications resources that it is configured to manage. The actual type of object created depends on the type of port, for each type of port object contains functionality specific to the resource it represents. After the port manager is started, it accepts requests for ports by other application software that has communications requirements.

Applications request Port objects by type and priority. When a request for a port is received, the PortManager finds a port of the specified type that is not currently in use and returns a reference to the port object to the requester. If all instances of a requested type of port are in use, a timeout value supplied by the requester is used to determine how long the requester is willing to wait for a port to become available. In the event there are two or more requesters waiting for a port to become available, the priority is used to determine which requester gets the next available port.

Once a port is acquired, it is accessed directly by its user to perform functionality specific to the type of port, such as connecting to a remote modem and/or sending and receiving bytes.

After a requester has finished using a port, it releases the port back to the PortManager. The port manager has a background process that reclaims ports as necessary if the user of a port does not release it.

## **2.3 Device Protocol Handlers**

Application objects known as device protocol handlers are provided as a high level interface to the FMS subsystem for specific device control. These protocol handlers are coded to communicate with a specific device type. Handlers for the following field devices are included in this design. Handlers for other device types can be added as needed.

### **Dynamic Message Signs (DMS)**

- FP9500
- FP2001
- FP1001
- TS3001
- Sylvia
- Display Solutions
- Addco

### **Highway Advisory Radio (HAR) [Future Release]**

- Information Station Specialists (ISS) AP55

### **SHAZAM [Future Release]**

- Viking RC2A remote on/off controller

Each protocol handler provides methods used by application programs to perform specific functions supported by the device targeted by the protocol handler. For example, a typical DMS protocol handler has methods to set a message, blank the sign, reset, and poll the DMS.

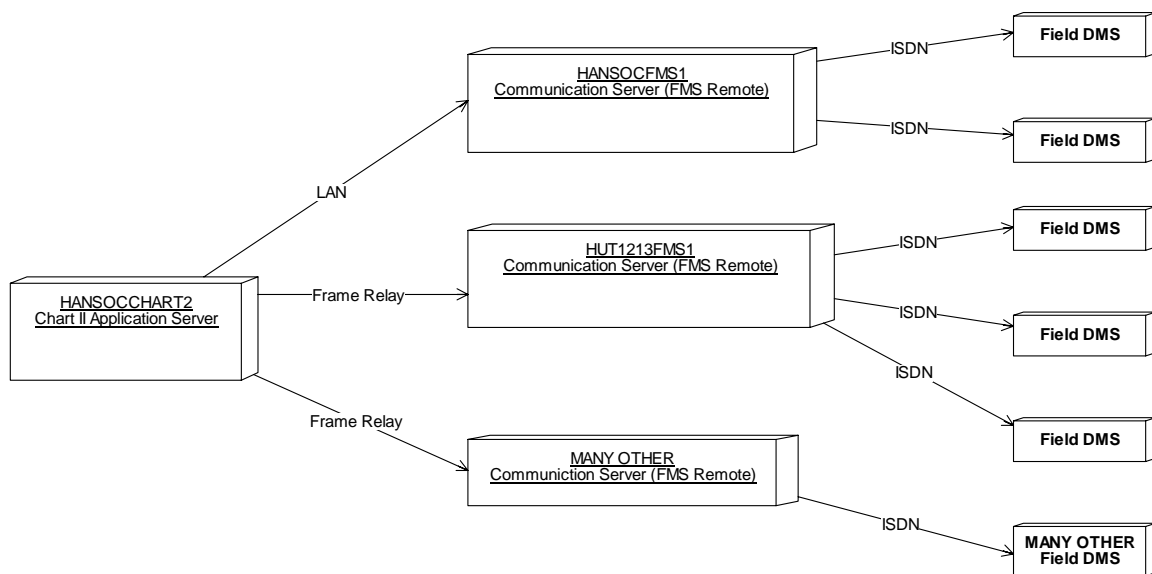
Device protocol handlers do not store device status or configuration. They only provide an encapsulation of the device protocol and act as a utility for higher-level applications that provide device control to an end user. The protocol handlers are provided a Port object through which they communicate with the device to fulfill a request.

## 2.4 CHART II Interface

The CHART II interface to the FMS subsystem is provided through device protocol handlers. CHART II provides application objects that model field devices and track their current configuration and status. These CHART II device objects construct a device protocol handler specific to the device type which the device object models and uses this protocol handler object when field communications are needed.

## 2.5 High Availability

The FMS software architecture allows for high availability of communications services through the deployment of many distributed PortManager objects. The FMS R1B1 hardware deployment lends itself to this wide distribution.



**Figure 1. FMS R1B1 Hardware Deployment (Deployment Diagram)**

Deployment of a PortManager object onto each Communication Server provides a high degree of redundancy in the system, for each PortManager provides the same basic service. Software that requires a PortManager has a choice from many PortManagers distributed throughout the system. One or many failures within the communications infrastructure (which includes full-scale communication server failure, ISDN port failure, frame relay circuit failure, and ISDN circuit failure) can be tolerated by the system as a whole.

Because some device communications (such as fixed DMS ISDN communications) benefit from cost savings through the use of a specific Communications Server close to the field device, the software that uses PortManagers can allow a preferred server to be specified, as well as one or more alternate servers. Under normal circumstances the preferred server would be used, however in the case where the preferred server is not available, an alternate server can be used to allow communications to proceed. A PortLocator utility class exists in this design to provide this functionality to users of FMS, namely CHART II device objects. This utility class provides a framework for a future enhancement to perform automated alternate server determination.

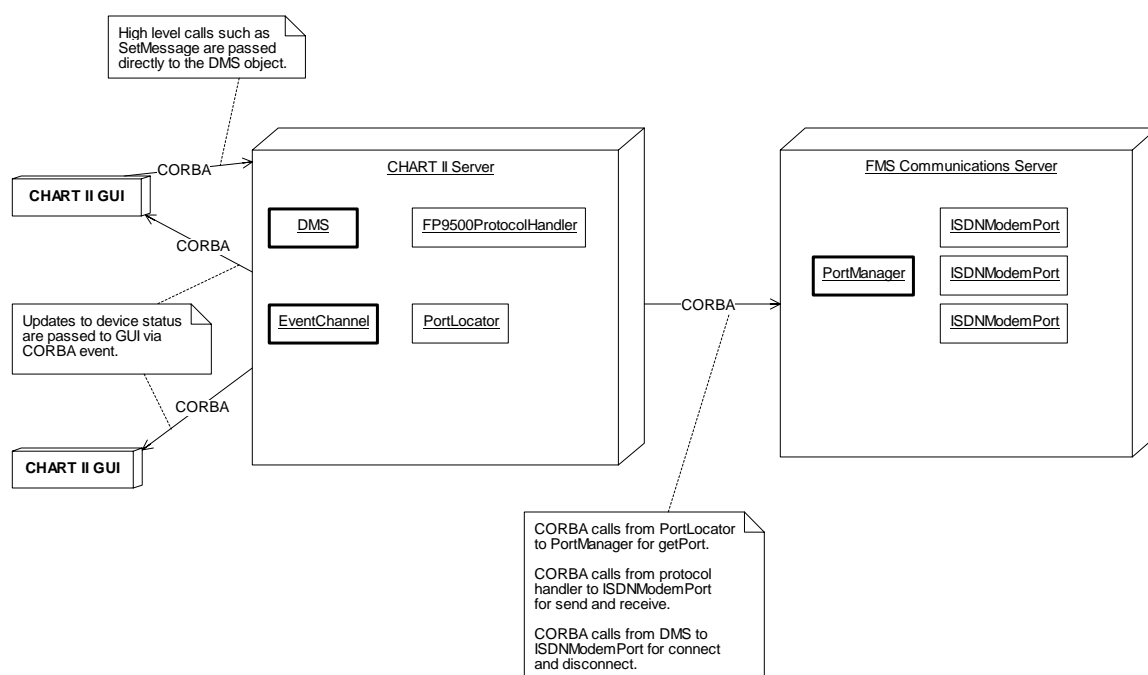
## 2.5.1 Device object deployment

Deployment options also exist with the CHART II objects that provide device control.

### 2.5.1.1 Lower Bandwidth CHART II Device Objects

With this deployment option, the application that uses protocol handlers is deployed on a PC other than an FMS Communications Server. The communications resources provided by the Communications Server are accessed remotely over a network via CORBA.

By deploying the device control software and the communications software on different physical PCs, the device control software is more tolerant to failures such as catastrophic communication server failures and frame relay failures.



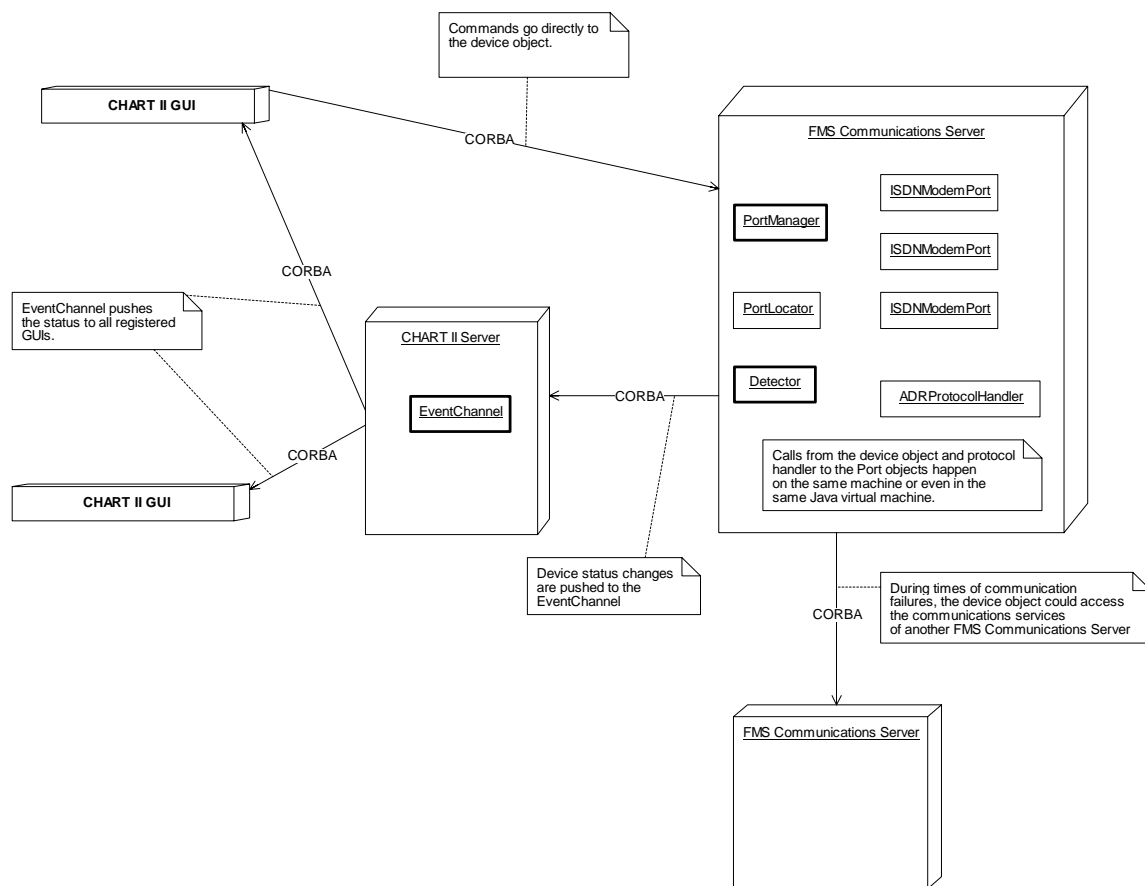
**Figure 2. Deployment for Lower Bandwidth Device Objects (Deployment Diagram)**

### 2.5.1.2 Higher Bandwidth CHART II Device Objects

Some field devices require frequent communications with a large amount of data. The cost of the network bandwidth required to control such a device using the PortManager objects in a distributed deployment may outweigh the benefits of the high degree of availability. In this case, one may choose to deploy the device control and protocol handler objects on the Communications Server itself to lower the network bandwidth usage.

Under this type of deployment, certain failure conditions would allow the use of the distributed nature of FMS to provide high availability to device control, while others would not. For example, if the Communications Server PC should fail completely, the device control software would become unavailable along with the PortManager object. If, however, there were a communications hardware or telecommunications infrastructure failure at the Communications

Server, the device control software could utilize a remote PortManager for its communications requirements instead of its local PortManager, incurring a higher bandwidth cost only during this time of failure.



**Figure 3. Deployment for Higher Bandwidth Device Objects  
(Deployment Diagram)**

## 2.6 Failures and Health Status

Failures are conveyed back to users of PortManager, Port, and protocol handler objects through exceptions. These exceptions contain detailed debugging information as well as information suitable for display to a user. The error messages provide details such as reason for a connection failure (such as busy signal) and protocol specific errors such as checksum failures, etc.

The FMS Communications Server software implements the Service interface defined by CHART II. This interface provides basic functionality useful for monitoring remote software processes. The implementation of this interface allows a system administrator to determine the health of the communications server software.

In addition to the generic health check provided by the Communications Server software, the PortManager objects provide a method that can be called to check the status of the ports that it manages. Port objects maintain a status that includes the number of connections attempted, the number of failed connections, and the number of input/output (IO) errors that have occurred on

the port. An application can use this information to allow a system administrator to determine the health of individual ports on a Communications Server.

## 2.7 Database Usage

The FMS Communications Server uses a database for configuration and object persistence. Each Port object served by the PortManager has entries in a database table providing configuration information such as a unique identifier and port name. Each Port entry also contains runtime statistics, such as number of uses and failure information.

Because the information being persisted to the database during runtime is not mission critical, the PortManager may periodically persist the Port objects instead of Port objects persisting themselves upon each operation. The exact approach to persistence will be determined in the detailed design.

Higher-level services such as the CHART II DMS service also only use the database for persistence and can also run on an Access database. It should be noted, however, that minor code changes would be needed to the CHART II software to make the database code portable between Oracle and Access.

## 2.8 ITS National Standards Approach

The FMS subsystem is designed to be compliant with the current ITS national standards in both the Center-to-Center and Center-to-Field requirements. The Center-to-Center requirements are met due to the fact that the interface supplied by FMS is CORBA, which is one of two approved methods of communication between ITS software components by the NTCIP Center-to-Center committee.

Center-to-Field compliance is met within the device protocol handlers. This design supports the addition of NTCIP compliant devices to the system through the addition of NTCIP device protocol handlers. For example, when an NTCIP DMS is acquired by SHA, a DMS NTCIP Protocol handler can be added that can be used to provide communications to any NTCIP compliant DMS.

## 2.9 R1B1 Problem Resolution

This design alleviates many problems that were present in the R1B1 release of FMS due to the elimination of the FMS Agent software layer:

- **MS Access Database** –Due to the fact that this design relies on the database only for start-up configuration information and runtime persistence of communications statistics (and not inter-process communications), the MS Access database will be sufficient for the FMS communications service. See the section on Database Usage above for more discussion.
- **Device configuration** – Since FMS does not store device configurations, the problems that caused a new device record to be added to FMS for configuration changes coupled with removal of the old device configuration are absent. Many configuration activities will require no FMS involvement.
- **Polling Interval/Coordination** – The specification of a polling interval as a polling cycle and the number of cycles is eliminated. Polling intervals will be specified in hours, minutes,

and seconds. Polling will be done by the device objects that can coordinate polling with commands that are originated by the user.

- **Conflict Request Error** – Because device control is performed through FMS in a synchronous manner, errors that occurred due to timeouts, mismatched and lost traps, and overlapping requests are eliminated.
- **Error Reporting** – Improved error reporting is provided through the use of exceptions that are thrown directly back to the CHART II software when failures occur.
- **Field Deployment** – This design allows a distributed deployment of code that is generic and not tied to high-level application logic. This minimizes the need for re-deployment to the remote FMS PCs.

## 2.10 Future Enhancements

This design provides a framework that is easily extensible. The current design covers basic communication services needed for DMS control. This section shows how upcoming enhancements fit into the architecture and also discusses future enhancements that may or may not be needed.

### 2.10.1 Currently Planned Enhancements

The following enhancements are currently planned and are present in this design:

1. **Telephony Port support:** A new type of port object will be added to support ports on a telephony card that allow DTMF and voice to be sent programmatically over a telephone line. This support will enable control of HAR and SHAZAM devices via FMS.
2. **HAR protocol handler:** A protocol handler will be developed to support the ISS AP55 HAR. This protocol handler will make use of the services provided by a Telephony Port object to program the HAR.
3. **SHAZAM protocol handler:** A protocol handler will be developed to support the Viking RC2A on/off controller, which is used to control a SHAZAM. This protocol handler will make use of the services provided by a Telephony Port to send DTMF to turn on or turn off the beacons on a SHAZAM.

## 2.10.2 Other Enhancements

The following enhancements lend to the maintainability of the FMS subsystem and also discuss how support for other device types can be added:

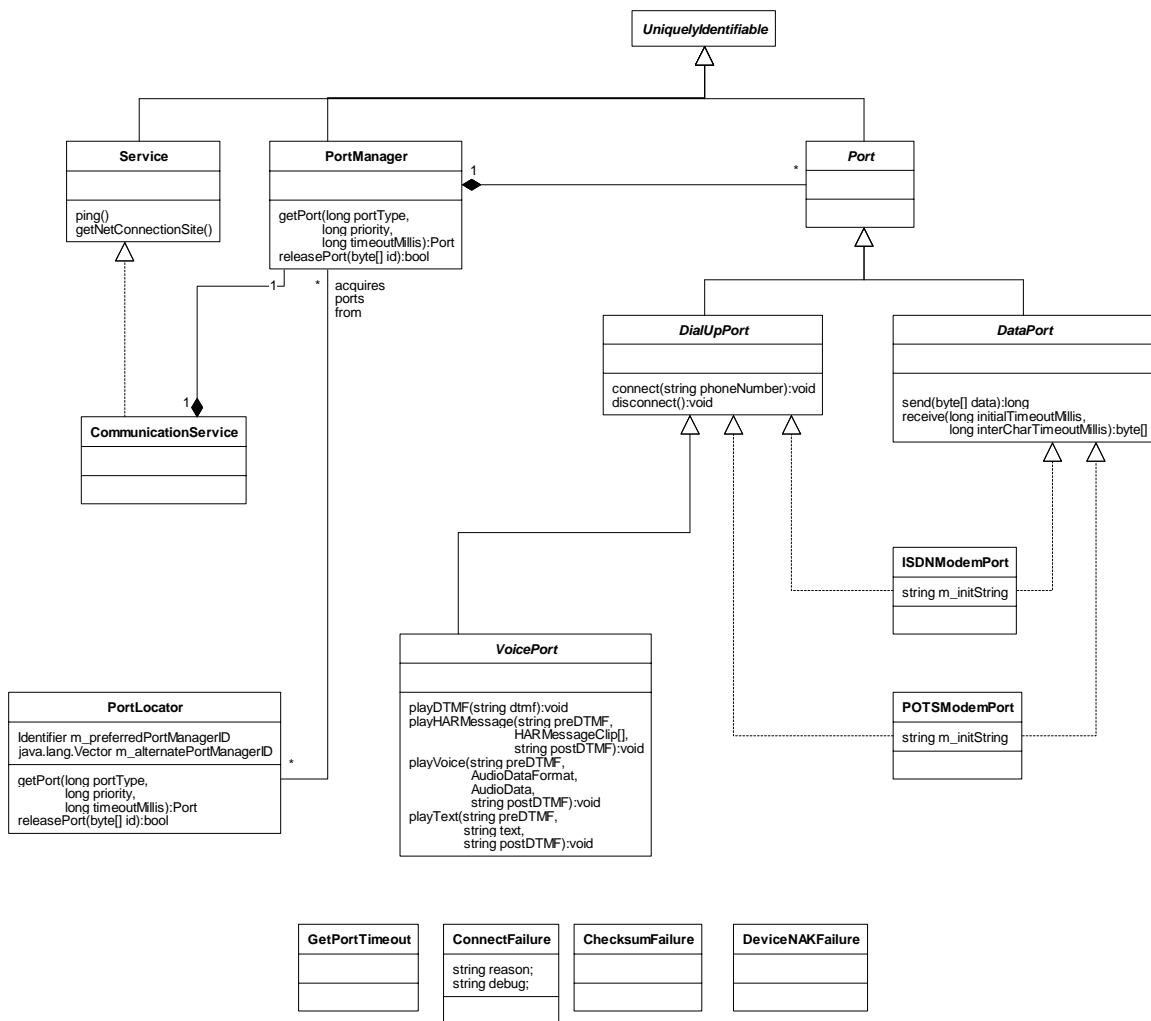
1. **Event based failure notification:** The PortManager object could register with a CORBA event channel and automatically send events when a port experiences errors. A GUI program could receive these events and create alarms, etc. that signal administrators of a possible problem.
2. **Remote Port Configuration:** Methods could be added to the PortManager to allow ports to be added, removed, disabled and enabled. This feature could be used to disable faulty ports until a technician fixes them or to set the configuration of the PortManager via a CHART II workstation.
3. **Support for dedicated port:** Methods could be added to the PortManager to request a specific port of a given type instead of the first available port. This would be handy in the lab environment that uses ISDN simulators, and would also be needed for devices that asynchronously report their status through dial-in.
4. **Support for dial in devices/synchronous receive:** A variation of the connect method could be supplied in a port that allows a listener object to be passed. Applications could configure a modem for auto answer and they would be passed data from the port asynchronously via their listener object when a device dials in and begins sending data.
5. **Additional Device Support:** Support for additional devices such as detectors can be added through the protocol handler framework included in this design.

## 3 Models

The following sections provide models and diagrams that show the high level design of the FMS R1B2 software. This section contains class diagrams to show software objects that comprise the system and their relationships between each other. Sequence diagrams are provided to show how objects interact to accomplish specific uses of the system.

### 3.1 Class Diagrams

#### 3.1.1 FMSSubsystem (Class Diagram)



**Figure 4. FMSSubsystem (Class Diagram)**

#### **3.1.1.1 ChecksumFailure (Class)**

This exception is used by ProtocolHandlers when the response from a field device contains a checksum error or when the device indicates that a packet sent to the device contains a checksum error.

#### **3.1.1.2 ConnectFailure (Class)**

This class is an exception thrown by Port objects when an attempt to connect to a remote device fails. This exception contains explicit failure information that is returned by the connection device, such as a modem response, etc.

#### **3.1.1.3 DataPort (Class)**

A DataPort is a port that allows binary data to be sent and received. Ports of this type support a receive method that allows a chunk of all available data to be received. This method prevents callers from having to issue many *receive calls* to parse a device response. Instead, this *receive call* returns all available data received within the timeout parameters. The caller can then parse the data on their side. Using this mechanism, device command and response should require only one call to send and one call to receive.

#### **3.1.1.4 ISDNModemPort (Class)**

This class is a port that provides access to an ISDN modem.

#### **3.1.1.5 DeviceNAKFailure (Class)**

This class is an exception thrown by protocol handler objects when a device responds to a request with a NAK. This class serves as an example of the use of exceptions by protocol handlers to provide specific failure information.

#### **3.1.1.6 DialUpPort (Class)**

A dial-up port is a port that is used to connect dial-up devices and requires a phone number to be passed at connection time.

#### **3.1.1.7 GetPortTimeout (Class)**

This class is an exception that is thrown by a PortManager when a request to acquire a port of a given type cannot be fulfilled within the timeout specified.

#### **3.1.1.8 CommunicationService (Class)**

This class is a high level place holder used to convey the fact that the FMS Communications Service will implement the Service interface defined in CHART II and that this application will serve one PortManager object.

#### **3.1.1.9 Port (Class)**

A Port is a CORBA interface that models a physical communications port. This interface is used as a base derivation point for subclasses that define specific types of ports.

#### **3.1.1.10 PortManager (Class)**

A PortManager is a software object that manages access to physical communications ports on a computer. The port manager allows ports to be requested by type and priority. When the demand for a specific type of port is greater than the supply, the PortManager queues the requests using priority. The PortManager also allows a timeout to be specified to indicate the amount of time the caller is willing to wait for a port to become available.

#### **3.1.1.11 POTSModemPort (Class)**

This class is a port that provides access to a POTS modem.

#### **3.1.1.12 Service (Class)**

This interface is defined in CHART II and is implemented by CHART II service executables. It provides a common interface that allows a system monitor to periodically check the health of a service.

#### **3.1.1.13 UniquelyIdentifiable (Class)**

This interface is implemented by classes whose instances have a unique identifier that is guaranteed not to match the identifier of any other uniquely identifiable objects in the system.

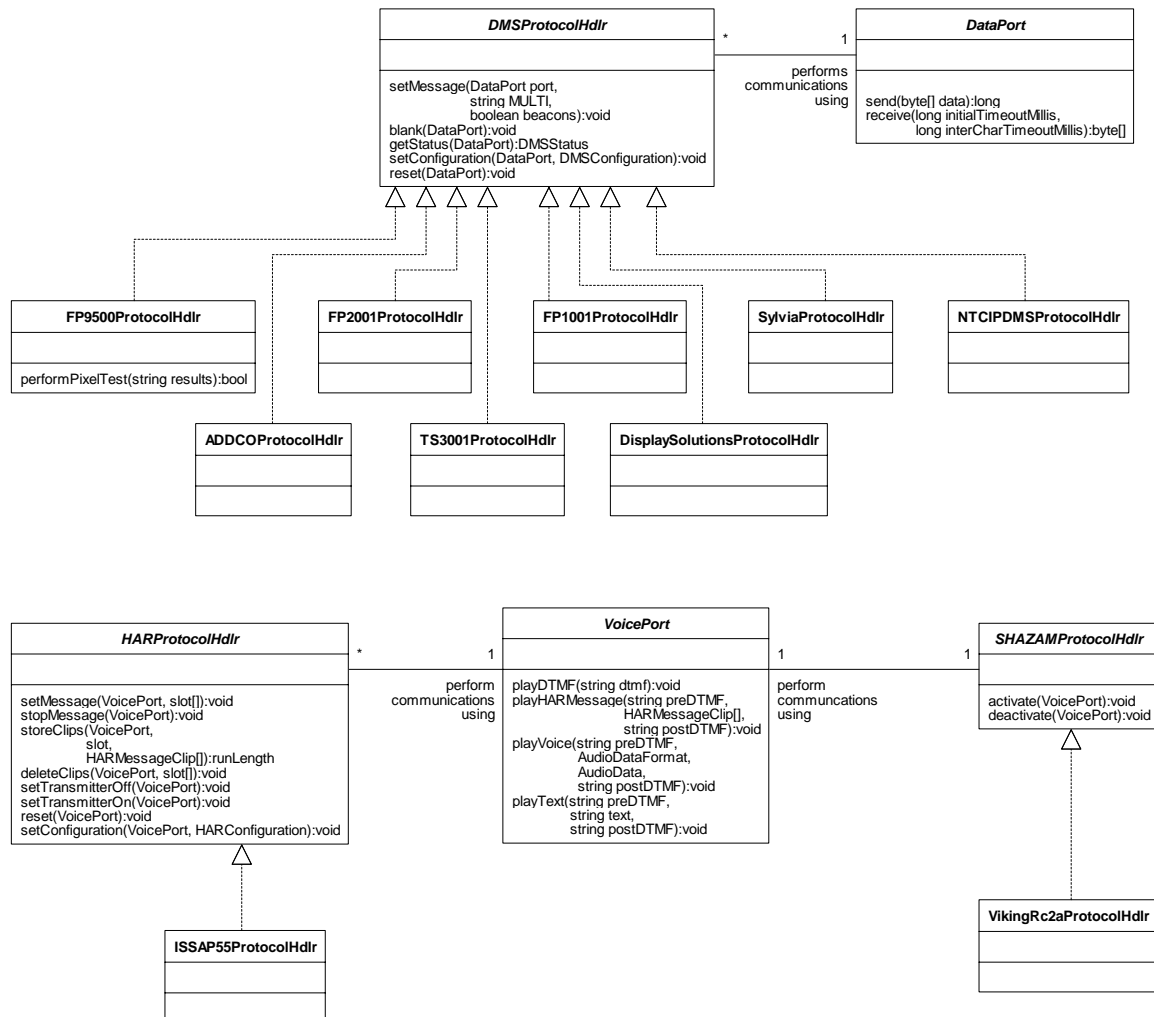
#### **3.1.1.14 VoicePort (Class)**

This class is a port that provides access to a port on a telephony card. While this port provides methods that may be used for low-level control, such as playing DTMF digits and playing a wave file, it also provides high level methods that bundle DTMF with playback. These high level methods are useful for controlling devices that require a command to be entered with DTMF and then have voice played. Bundling the command allows the method to prepare to play the voice prior to sending the DTMF command. This is especially useful when the voice preparation could take several seconds, such as when text is converted to speech.

#### **3.1.1.15 PortLocator (Class)**

The PortLocator is a utility class that helps one to utilize the fault tolerance provided by the deployment of many PortManagers. The PortLocator is initialized by specifying a preferred PortManager and optionally one or more alternate PortManagers. When asked to acquire a port, the PortLocator first attempts to acquire a port from the preferred PortManager and falls back to alternate PortManager objects when faults occur. The PortLocator can also be set to determine the fallback action (if any) if the preferred PortManager does not have any ports currently available. For example, the PortLocator could be set to wait a maximum of  $x$  seconds for a port and then attempt to acquire a port from an alternate PortManager.

## 3.1.2 FMSProtocolHandlers (Class Diagram)



**Figure 5. FMSProtocolHandlers (Class Diagram)**

### 3.1.2.1 ADDCOPProtocolHdlr (Class)

This protocol handler implements the protocol used to command an ADDCO DMS.

### 3.1.2.2 DMSProtocolHdlr (Class)

This interface defines the methods that must be supported by DMS protocol handlers. Note: some handlers support methods in addition to these standard methods.

### 3.1.2.3 FP1001ProtocolHdlr (Class)

This protocol handler implements the protocol used to command an FP1001 DMS.

#### **3.1.2.4 DisplaySolutionsProtocolHdlr (Class)**

This protocol handler implements the protocol used to command a Display Solutions DMS.

#### **3.1.2.5 FP2001ProtocolHdlr (Class)**

This protocol handler implements the protocol used to command an FP2001 DMS.

#### **3.1.2.6 FP9500ProtocolHdlr (Class)**

This protocol handler implements the protocol used to command an FP9500 DMS.

#### **3.1.2.7 SHAZAMProtocolHdlr (Class)**

This interface specifies the methods that must be implemented by protocol handlers that implement the protocol to command a SHAZAM device. The methods provide a way to enable and disable the flashers that exist on a SHAZAM.

#### **3.1.2.8 HARProtocolHdlr (Class)**

This interface specifies the methods that must be implemented by a protocol handler used to program a HAR. Methods are provided to allow messages to be downloaded and played on the HAR and to allow for maintenance activities, such as toggling the transmitter on and off.

#### **3.1.2.9 ISSAP55ProtocolHdlr (Class)**

This protocol handler implements the protocol used to program an ISS AP55 HAR.

#### **3.1.2.10 NTCIPDMSProtocolHdlr (Class)**

This protocol handler implement the protocol used to command an NTCIP compliant DMS. This class is shown to illustrate how NTCIP center to field compliance is addressed by this design.

#### **3.1.2.11 SylviaProtocolHdlr (Class)**

This protocol handler implements the protocol used to command a Sylvia DMS.

#### **3.1.2.12 TS3001ProtocolHdlr (Class)**

This protocol handler implements the protocol used to command an Telespot DMS.

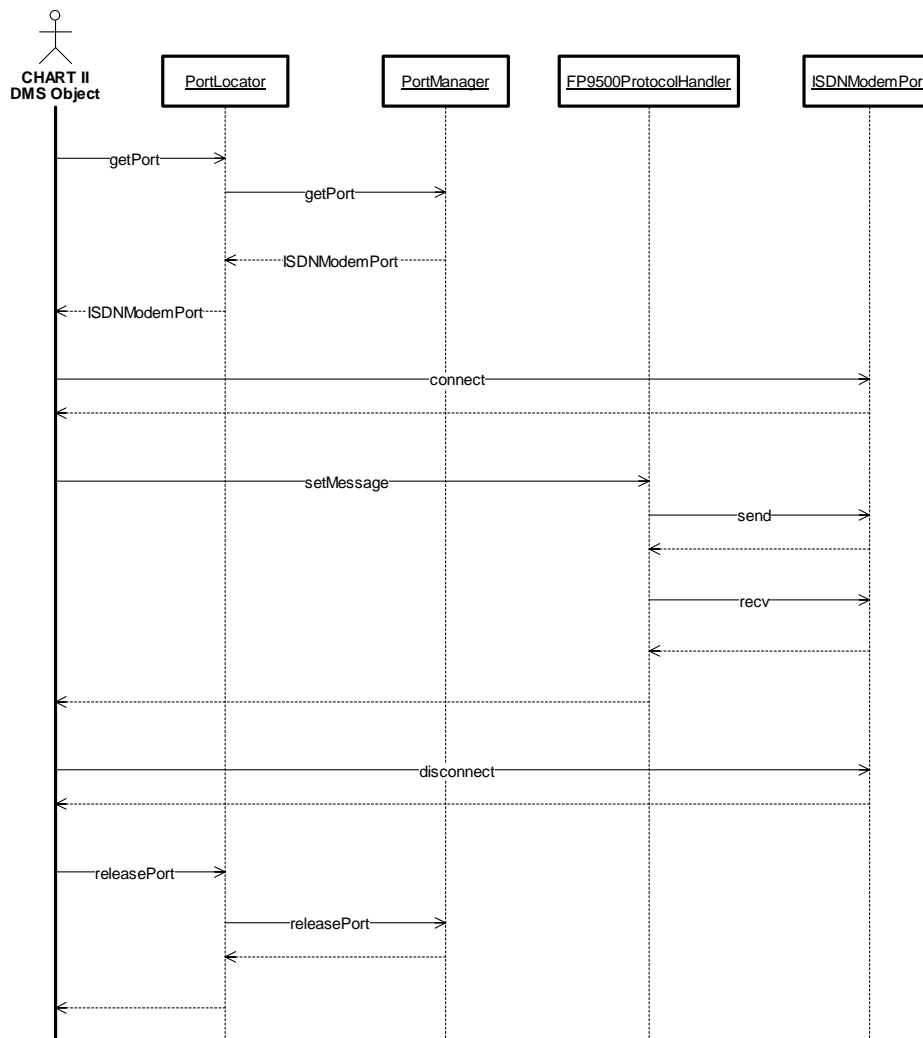
#### **3.1.2.13 VikingRc2aProtocolHdlr (Class)**

This class implements a protocol handler that provides the protocol to command a Viking RC2A on/off controller. This controller is used to turn the beacons on and off on the flashing advisory signs known by the SHA as SHAZAMs.

## 3.2 Sequence Diagrams

### 3.2.1 Typical FMS Usage – Set DMS Message

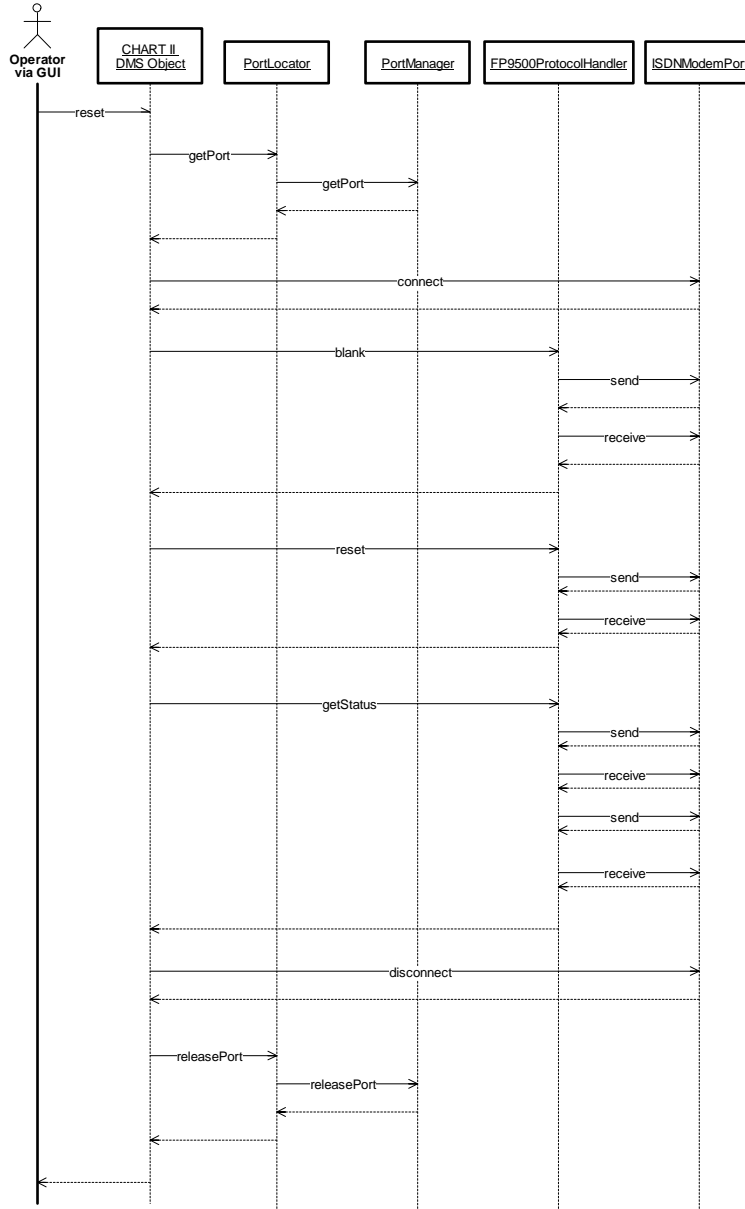
This sequence shows the basic object interactions that occur to perform device communications via FMS, using the task of setting a message on a DMS as an example. The device object initializes communications by first using the PortLocator utility to acquire a port of a specific type. The PortLocator obtains a port from its preferred PortManager and returns it to the device object. The device object establishes a connection via the port object and then communicates with field device using the protocol handler.



**Figure 6. Typical FMS Usage – Set DMS Message (Sequence Diagram)**

### 3.2.2 Multiple Commands on Single Connection

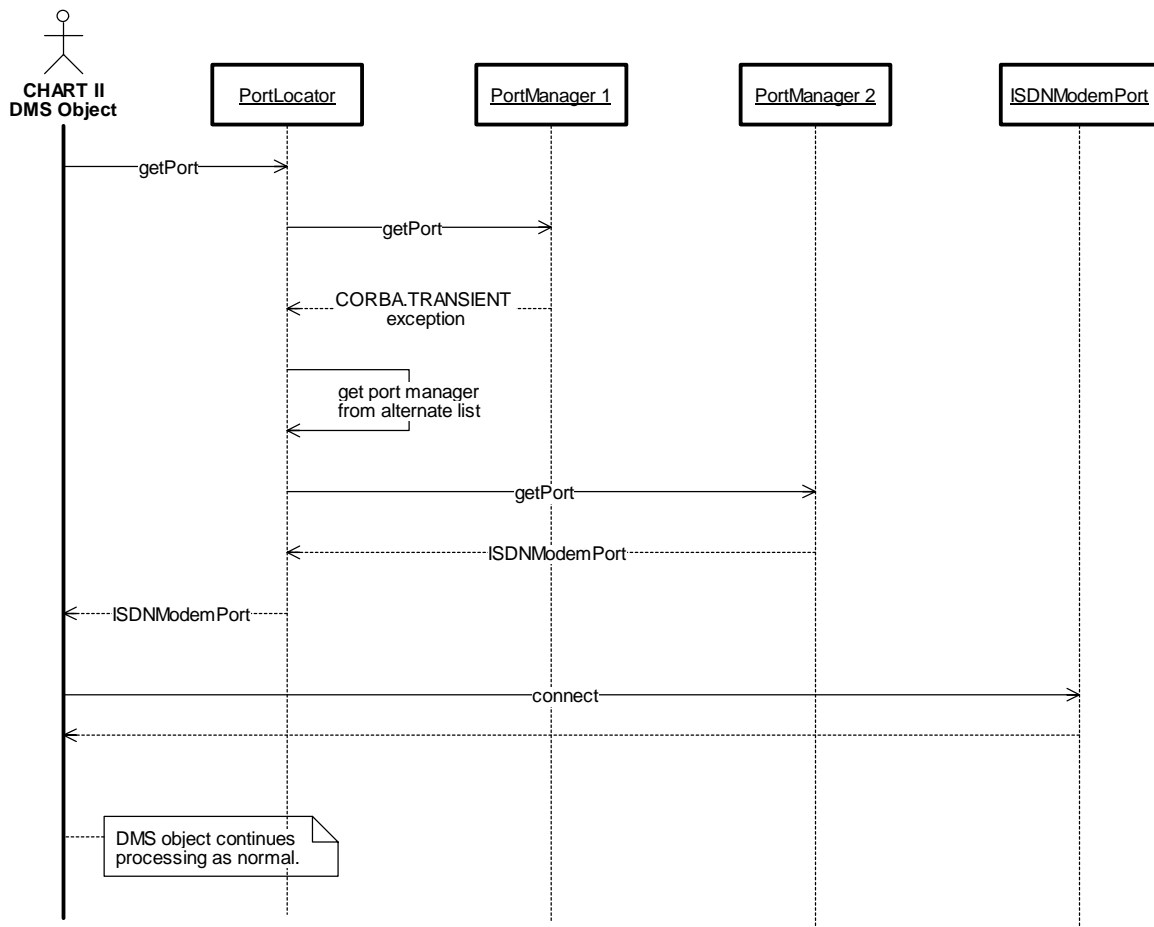
This diagram shows usage of the FMS subsystem for a high level device command that requires sending many commands to the field device. Such is the case in resetting a DMS, which requires commands to the DMS to blank it, reset it, and then get its status. Because of the fine grained control given to the user of FMS, a single connection can be held open by the object that sends multiple commands to avoid the time required to reconnect. Note that the PortManager contains a mechanism to prevent an application from grabbing a port and attempting to hold onto it as their own. See the section on Port Leak Prevention.



**Figure 7. Multiple Commands On Single Connection (Sequence Diagram)**

### 3.2.3 Fault Tolerance Scenario

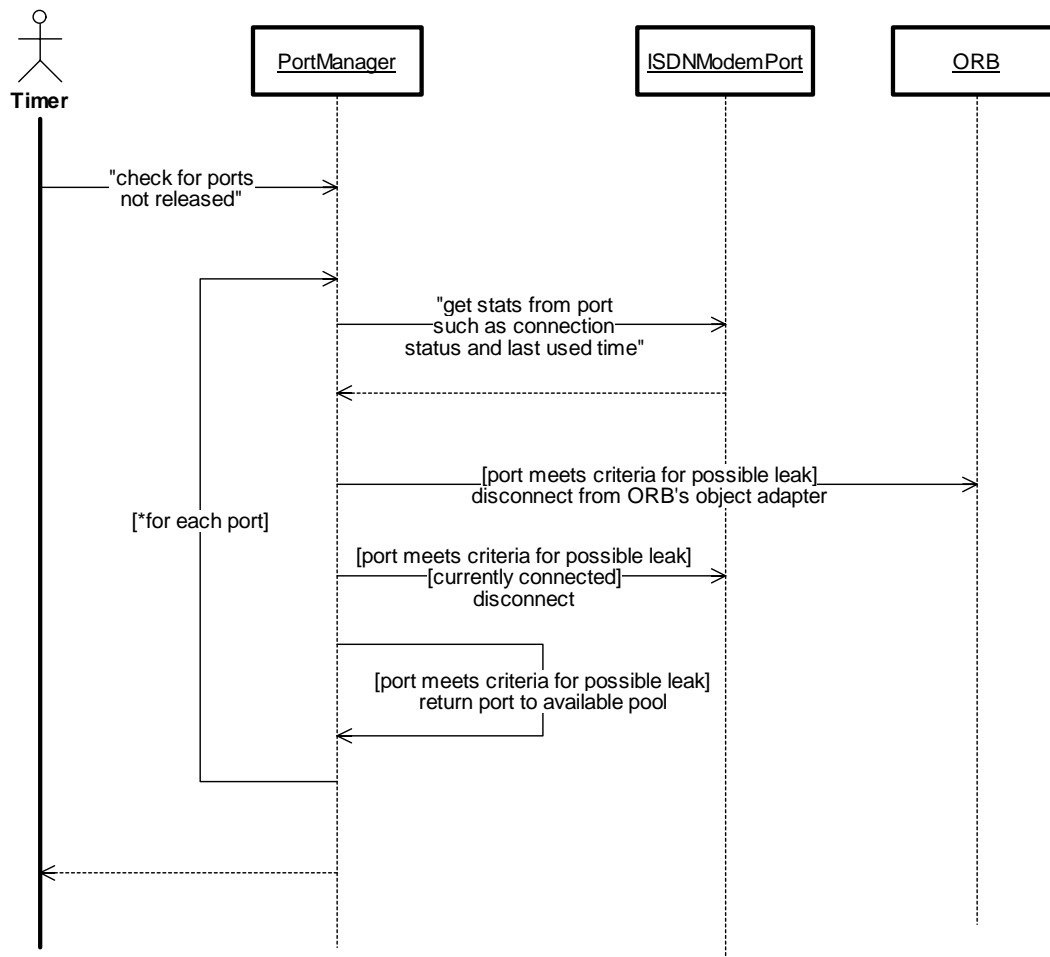
This diagram shows that the PortLocator contains the logic for fault tolerance capabilities. When asked to get a port, the PortLocator first attempts to get a port from the preferred PortManager, but may fall back to alternate PortManagers when a failure occurs. The user of the PortLocator does not need to worry about this logic and performs the normal processing on whatever port is returned by the PortLocator.



**Figure 8. Fault Tolerance Scenario (Sequence Diagram)**

### 3.2.4 Port Leak Prevention

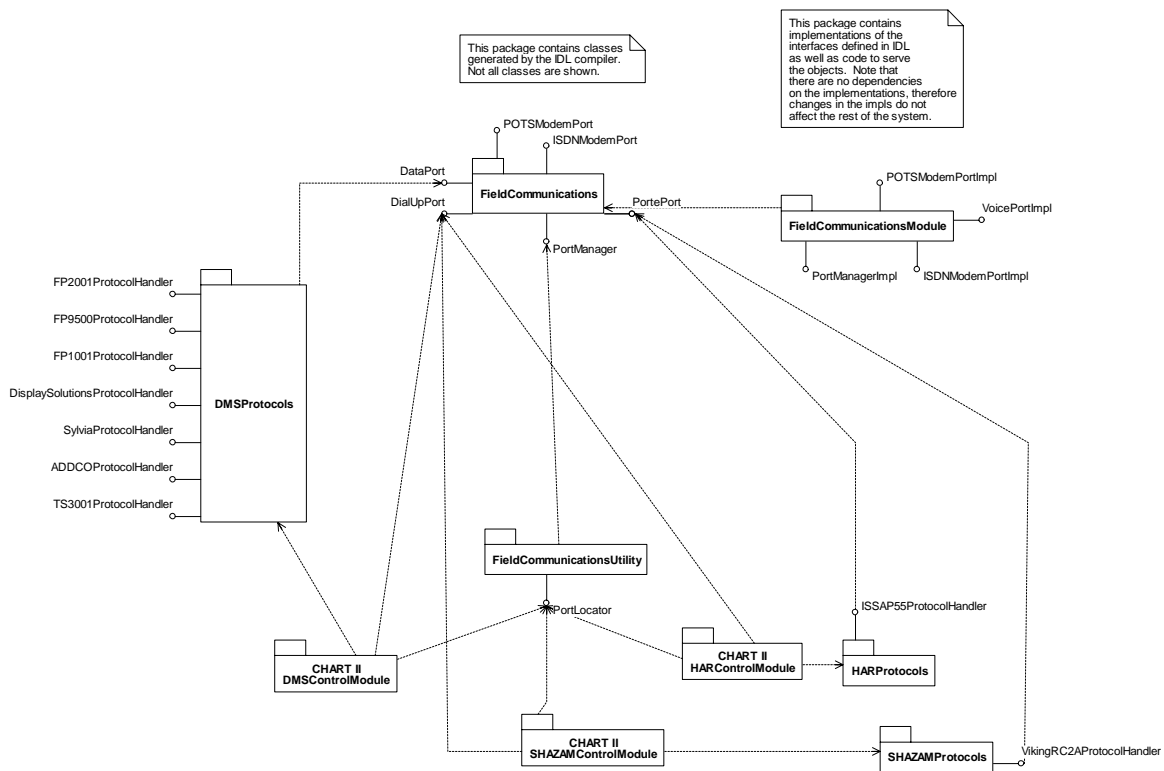
The PortManager allows ports to be acquired and released by its users. The PortManager has a mechanism to reclaim ports that have not been released in a timely manner. The PortManager periodically checks each of its ports and evaluates them for criteria that would indicate the port's user is finished with the port but didn't release the port. When this occurs, the port is disconnected from the ORB, making it unavailable for use by its current holder. The port is then returned to the pool of free ports. The next time the port is given out, it is reconnected to the ORB and the new object reference is passed to the requestor. The use of transient CORBA objects allows the PortManager to ensure that only one requester ever has a valid reference to each Port object. A holder of a reference to a port that has been reclaimed will receive a CORBA exception if they try to use the port.



**Figure 9. FMSSubsystem: Port Leak Prevention (Sequence Diagram)**

## 4 Packaging

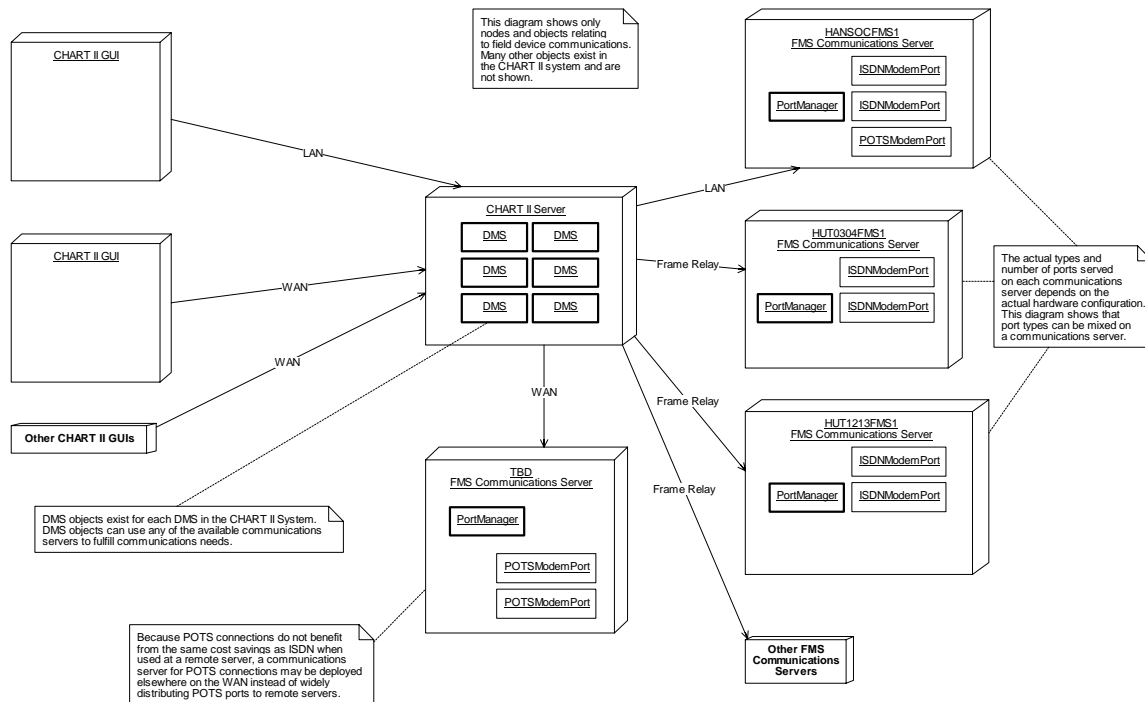
This diagram shows the software packages for the FMS subsystem as well as CHART II packages that will interface with FMS. Dependencies indicate how changes to an interface will affect other software in the system. A dependency line indicates that if the target interface is changed, the dependent software will need to be reviewed for possible changes as well.



**Figure 10. FMSPackaging (Class Diagram)**

## 5 Deployment

This deployment shows the probable deployment of the FMS objects for R1B2. The actual number and types of ports is not accurate on this diagram and is shown as an example of the ability for a PortManager to manage ports of different types. ISDN modems have already been deployed at remote sites for communications cost savings. POTS modems will also be deployed at sites where they can benefit from a maximum cost savings (i.e., by LATA).



**Figure 11. FMSDeployment (Deployment Diagram)**

## Bibliography

---

*CHART II Business Area Architecture Report*, document no. M361-BA-005R0, Computer Sciences Corporation and P.B. Farradyne, Inc., April 28, 2000

*CHART II System Requirements Specification Release 1 Build 2*, document no. M361-RS-002R1, Computer Sciences Corporation and P.B. Farradyne, Inc.

*The Common Object Request Broker: Architecture and Specification*, Revision 2.3.1, OMG Document 99-10-07

Martin Fowler and Kendall Scott, *UML Distilled*, Addison-Wesley, 1997

*TELE-SPOT 3001 Sign Controller Communications Protocol*, document no. 750208-040 v2.3, T-S Display Systems Inc., 1995

*Functional Specification for FP9500ND – MDDOT Display Control System*, document no. A316111-080 Rev. A6, MARK IV Industries Ltd., 1998.

*Maintenance Manual for the FP1001 Display Controller*, document no. 316000-443 Rev. E, Ferranti-Packard Displays, 1987

*FP2001 Display Controller Application Guide*, document no. A317875-012 Rev. 8, F-P Electronics, 1991

*Engineering Specification - Brick Sign Communications Protocol*, Rev. 1, ADDCO Inc., 1999.

*PCMS Protocol version 4*, document number 32000-150 Rev. 5, Display Solutions, 2000

*BSC Protocol Specification (Data Link Protocol Layer)*, v. 1.3, Fiberoptic Display Systems Inc., 1996

*Sylvia Variable Message Sign, Command Set 9403-1*, v. 1.4, Fiberoptic Display Systems Inc., 1996

*2.5 Mile AM Travelers Information Station Instruction Manual For: Maryland State Highway Administration, Information Station Specialists.*

*Technical Practice RC-2A Remote Touch-Tone On/Off Industrial Controller*, Viking Electronics Inc., August 1993.

# Acronyms

---

The following acronyms appear throughout this document:

API	Application Program Interface
CORBA	Common Object Request Broker Architecture
DBMS	Database Management System
DMS	Dynamic Message Sign
DTMF	Dual Tone Multiple Frequency
FMS	Field Management Station
GUI	Graphical User Interface
HAR	Highway Advisory Radio
IDL	Interface Definition Language
ITS	Intelligent Transportation Systems
LATA	Local Access and Transport Areas
NTCIP	National Transportation Communications for ITS Protocol
OMG	Object Management Group
ORB	Object Request Broker
POTS	Plain Old Telephone System
R1B2	Release 1, Build 2
UML	Unified Modeling Language

# Glossary

---

<b>Communications Server</b>	A PC outfitted with communications hardware and the FMS Communications Service software.
<b>DMS</b>	A Dynamic Message Sign that can be controlled by one Operations Center at a time.
<b>Graphical User Interface</b>	Part of a software application that provides a graphical interface to its user.
<b>HAR</b>	A Highway Advisory Radio which can be controlled by one Operations Center at a time.
<b>Operator</b>	A Chart II user that works at an Operations Center.
<b>Port</b>	A software object used to model a physical communications port.
<b>Port Manager</b>	A software object that manages access to one or more communications ports.
<b>Protocol Handler</b>	A software object that contains code that encapsulates the specific communications sequences required to command a field device.
<b>SHAZAM</b>	A device used to notify the traveling public of the broadcast of a HAR message.
<b>User</b>	A user is somebody who uses the CHART II system. A user can perform different operations in the system depending upon the roles they have been granted.

## **Appendix A: CORBA Information**

---

### **CORBA**

CORBA is an architecture specified by the Object Management Group (OMG) for distributed object oriented systems. The CORBA specification provides a language and platform independent way for object oriented client/server applications to interact. The CORBA specification includes an Object Request Broker (ORB) which is the middleware used to allow client/server relationships between objects. Using a vendor's implementation of an OMG ORB, software applications can transparently interact with software objects anywhere on the network without the application having to know the details of the network communications.

Interfaces to objects deployed in a CORBA system are specified using OMG Interface Definition Language (IDL). Applications written in a variety of languages or deployed on a variety of computing platforms can use the IDL to interact with the object, regardless of the language or computing platform used to implement the object.

### **CORBA Services**

The OMG CORBA specification includes specifications for application servers that provide basic functionality that is commonly needed by distributed object systems. While there are specifications for many such services, many services have not yet been implemented. Of the CORBA Services that are available, the CORBA Event Service and CORBA Trading Service are utilized in the CHART II system. A description of each of these services follows.

#### **CORBA Event Service**

The CORBA Event Service provides for a way to provide data updates within the system in a loosely coupled fashion. This loose coupling allows applications with data to share to pass the information via the event service without needing to have knowledge of others that are consuming the data.

Data passed through the event service is done using event channels. Many different types of events may be passed on a single event channel. Interested parties may become consumers on a given event channel and receive all events passed on the channel.

The CHART II system makes use of multiple event channels to allow event consumers to be more selective about the type of events they receive. Also, event channels of the same type may exist in multiple regions, allowing the CHART II system to be expandable and multi-regional. Event channels used in the CHART II system are published in the CORBA trading service to allow others to select which events they wish to consume.

#### **CORBA Trading Service**

The CORBA Trading Service is an online database of objects that exist in a distributed object system. Servers that have services to offer publish their objects in the trading service.

Applications that wish to use the services provided by a server can query the Trading Service to find objects based on their type or attributes.

CORBA Trading Services can be linked together into a federation. Queries done on single Trading Service can be made to cascade to all linked Trading Services as well. This feature allows Trading Services serving single regions to be linked together, providing seamless access to all objects in the system.

The CHART II System utilizes the CORBA Trading Service to allow the GUI to discover objects in the system with which it allows the user to interact. Using the linking capabilities of the Trading Service, the CHART II system can be distributed to multiple districts with the GUI still able to provide a unified view of the system to the users.